View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by TUT DPub

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ARVI SYRJÄNEN

TASK LEVEL ROBOT PROGRAMMING: BACKGROUND, METHODS AND CURRENT STATE

Bachelor of Science Thesis

Examiner: prof. José L. Martínez Lastra
23 November 2018

ABSTRACT

Arvi Syrjänen: Task level robot programming: Background, methods and current state
Tampere University of technology
Bachelor of Science Thesis, 33 pages
NOVEMBER 2018
Bachelor's Degree Programme in Automation Engineering
Major: Factory Automation
Examiner: Professor José L. Martínez Lastra

Keywords: task level, task-oriented, robot programming, history, automation, software, human-machine interface, multimodal interaction, programming by demonstration

The thesis is a literature review on task level robot programming. It has been researched from various point of views, starting from the history of robot programming in general to find the underlying motivations for more intuitive and abstract robot programming, which is followed by the methods and general structure of a task level programming environment. A survey on the state of the art is also presented in this paper, where currently available commercial robot programming systems and studies in literature are reviewed.

Task level robot programming makes robot programming more intuitive, faster and requires no programming expertise if done correctly. Small to medium sized enterprises with small batch sizes, frequently changing products and possibly underqualified staff with regard to programming skills would benefit most from intuitive human-robot interfaces by being enabled to access the production efficiency of robotics without programming expertise. Currently there are few commercial process specific robot programming systems, where robot programming is automatic and the user inputs only a task specification. Addition to the commercial systems, task level programming and intuitive human-machine interfaces such as multimodal-interaction or teaching by demonstration are emerging research topics.

PREFACE

Moving machines and especially those seemingly intelligent robots have always fascinated me. For this reason, it seemed natural and exciting for me to study robot environments that have capabilities of autonomous decision making in my bachelor's thesis. Before I started working on this thesis, I knew hardly anything at all about robot programming and writing this thesis has been a great learning opportunity for me.

I would like to express my gratitude towards my thesis director and examiner professor José L. Martínez Lastra for all the help I received from him during the research process.

Tampere, 23.11.2018

Arvi Syrjänen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ABB – Asea Brown Bover

AML – Automation Markup Language

CAD – Computer aided design

CAM – Computer aided manufacturing

HRI – Human-robot interaction

IBM – International Business Machines Corporation

IFR - International Federation of Robotics

KRL – KUKA Robot Language

KUKA – Keller und Knappich Augsburg

MMI – Multimodal interaction

PC – Personal Computer

PUMA – Programmable Universal Arm for Assembly

RAPID – Robot programming language of ABB robots

RPL – Robot programming language

SME – Small to medium sized enterprise

UI – User interface

VAL – Variable Assembly Language or VicArm Language

XML – Extensible Markup Language

1. INTRODUCTION

Motivation

The motivation for this thesis stems from the rising sales of robots, widening userbase of robots and the trend of mass personalization production coming along with the fourth industrial revolution (Wang et al. 2017). According to International Federation of Robotics (IFR), in 2016, industrial robot sales were 294 312 units, professional service robot sales were 59 706 units and service robots for personal and domestic use sales were 6,7 million units. While all robot sales experienced an overall growth in 2016, service robot sales grew faster than industrial robot sales, 24% versus 16% compared to 2015. According to the same authority, during 2017 industrial robot sales growth almost doubled to 31%. The accelerating growth of service robot sales implies that more people outside the industrial field are interacting with robots, while simultaneously small to medium sized enterprises (SME) are utilizing robots in their businesses increasingly (International Federation of Robotics 2018; Brunete et al. 2017). The average robot user is starting to be something else than an engineer or other experienced operator. Simultaneously, traditional players in manufacturing industries need to answer for the demand of individualized and quickly changing products, which requires fast reconfiguration of manufacturing equipment (Backhaus & Reinhart 2013). There clearly is a need to raise the level of abstraction in robot programming so that even wider audiences have possibilities to interact with them and reprogramming becomes more efficient. This has lead for intuitive robot interfaces receiving more and more attention in recent years (Ekvall & Kragic 2008).

Justification

The ability to make quick changes and reconfigurations to meet constantly changing market demands is one of the most important factors for a production system now and it will be emphasized even more in near future (Backhaus & Reinhart 2013). Higher level of abstraction in the programming of production robots makes teaching new tasks to robots faster, easier and requires less programming expertise from the user. High level of abstraction allows the robot programmer to be a process specialist without extensive programming skills. Task level programming is robot programming at the highest level of abstraction and this is the reason why it is researched in this thesis.

Research questions and scope

This thesis aims to find motivation for easier robot programming interfaces, to research task level programming methods and to review the current state of the art in task level

robot programming. The research question could be summarized as “What is task level robot programming, why and how has it been developed and what is its state now”. The scope of the paper is programming of industrial robotic manipulators one would encounter in a factory environment.

Structure of the document

The thesis consists of five parts. First, some background about robot programming will be provided. Next some history about robot programming will be reviewed, which serves as motivation towards higher level of abstraction in robot programming and task level robot programming. In chapter 4, the generally accepted methods and the basic structure of a task level programming environment will be reviewed. This is followed by a survey on the current state of the art in task level robot programming and other high-level programming methods. In the chapter 4.4, which discusses the state of the art, a lot of information regarding commercial products is gained from the manufacturers’ or software providers’ own material and industry related web-articles, not from peer reviewed scientific literature. Finally, conclusions will be assessed.

2. BACKGROUND

Robot programming is about teaching the robot's controller how its manipulator should complete job specific tasks, or more precisely, the motions the manipulator should make to complete these tasks. The robot usually has, at least for safety measures, some sensors in its work environment and the robot reacts to the sensor inputs according to its programming, e.g. when a person walks into the robot's working area, the robot stops all movement.

Programming a work task to a robot manipulator appears relatively simple and straight forward at first. The programmer tells the robot a set of desired goal states, defined by the location of the end-effector with one of the ways explained in the following subchapters. When the program is executed, the robot moves so that the end-effector reaches these goal states.

At first glance, robot program code reminds traditional computer programming code a lot, both having familiar for-loops and if-else statements. Robot control programs are written in manufacturer dependent programming languages, such as ABB's RAPID and KUKA's KRL. Even though languages from different manufacturers often look very similar to each other, there are semantic differences between them (Hägele et al. 2008). One important quality that distinguishes robot programming from traditional computer programming is the cyber-physical nature of automation systems; a bug in a program might cause serious real-world damage – economic losses or human injury. This and the fact that the users of robots are usually shop-floor workers in factories with little programming knowledge are the reasons why robot programming methods are a bit different from those of conventional computer programming.

Robot programming methods have been traditionally divided into online and offline programming, depending on *where* the person is during programming. Online programming happens at the robot's work area, with a line of sight and a physical contact with the manipulator. Offline programming happens away from the physical robot in a simulated environment on a normal PC. With offline programming, the robot system can be programmed and tested completely independently of the robot itself and the actual production line might not even be built at the time of programming.

2.1 Online programming

There are a couple of popular ways to do online programming. Teach-pendant programming and lead-through teaching. When programming with a teach pendant, the user operates the robot with a separate, usually a handheld, device visible in figure 1, which has control over each joint individually. The user can jog the manipulator to the desired positions and orientations and record those points. After necessary points are declared, the user replays the cycle created and visually confirms its validity or fixes it if it is unsatisfactory. (Yong et al. 1985)

Figure 1. *A teach pendant used in online programming (Motion Control Robotics 2017)*

Another way of programming the manipulator online is lead-through teaching. In lead-through teaching, the robot arm is manually dragged to the desired positions and orientations. By lead-through teaching it is possible to, instead of teaching points, teach the robot exact paths and this makes teaching continuous path motions (figure 2) easier, which are necessary for example in welding. The positions of the manipulator are recorded with preset intervals to get a digital model of the end-effectors path. (Deisenroth 1985)

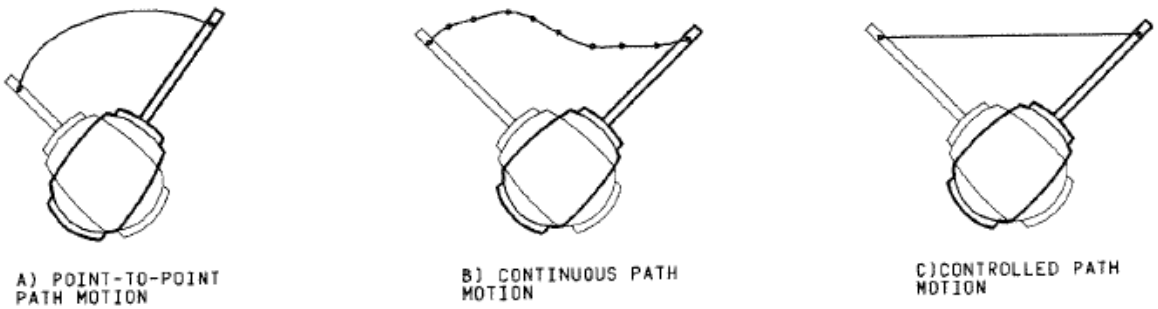

Figure 2. *The three movement types of the robot's end effector. (Deisenroth 1985, p. 353)*

2.2 Offline programming

In offline robot programming the baseline situation is that the user writes robot control code that controls the robot manipulator. Addition to just writing some code, offline programming utilizes a lot of 3D-models and simulations for visualizing the robot's move-

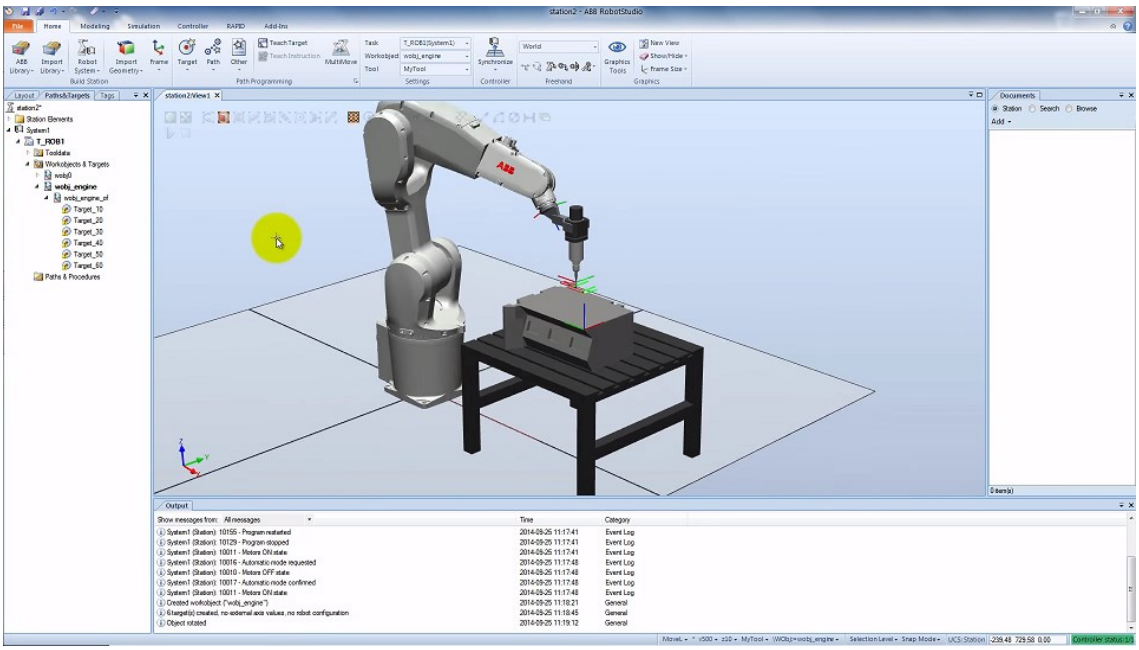

Figure 3. *User Interface (UI) of an offline programming environment RobotStudio by ABB (ABB 2017)*

ments and its environment. After coding, the program is verified in a visualization that lets the programmer to see how the code actually moves the robot. In most modern offline programming environments, such as ABB RobotStudio or Fanuc's RobotGuide, the manipulator can be virtually jogged inside the simulation to teach points and trajectories, just like in online programming.

3. EVOLUTION OF ROBOT PROGRAMMING METHODS AND ENVIRONMENTS

When robots first appeared in industry, the more popular choice of programming methods for the first decades was online programming due to its simplicity, intuitiveness, (Yong et al. 1985) and the underdeveloped state of offline programming at the time. Online programming is often seen the easier option which requires little computer skills, but as the task at hand becomes more complex, the time teaching it manually increases. Since online programming requires production to be halted for the time of the programming, nowadays off-line programming becomes very quickly the more economically feasible choice. Offline programming also removes the operator from the immediate proximity of the production machinery, which contributes to personnel safety and allows the programmer to be physically anywhere.

Even in the start of the 90's offline programming was not yet practical enough and according to Fuller (1991, p. 276) programming anything but a cartesian-coordinates robot was difficult at best. A 1995 article in "The Industrial Robot; Bedford"-magazine commented on the problems of offline programming stating that since the early days of offline programming, 1989 or so [sic], it has not yet delivered quite what has been promised. The robots were sold with the promise that they are delivered fully programmed to the production line. However, the robot models supplied for simulation did not represent real-world robots precisely enough and this often resulted in a hybrid solution with offline programming and costly online programming to correct manually the offline created programs to work with the robots on-site.

At some point at the turn of the millenium, offline programming went past online programming as the more popular method as the online programming environments developed. In 1999 Deisenroth and Krishnan claim that online programming is still the more widely used method of robot programming, but in 2008 Hägele et al. claim using online programming to be unusual in industry except to verify and fine-tune programs created offline. The advances in offline programming are greatly associated with progress in 3D-simulation technology and personal computer performance (Ahrens 2001).

Especially in the early days of offline programming it required advanced programming skills to and the programming interfaces were not very user-friendly. Before easier user-

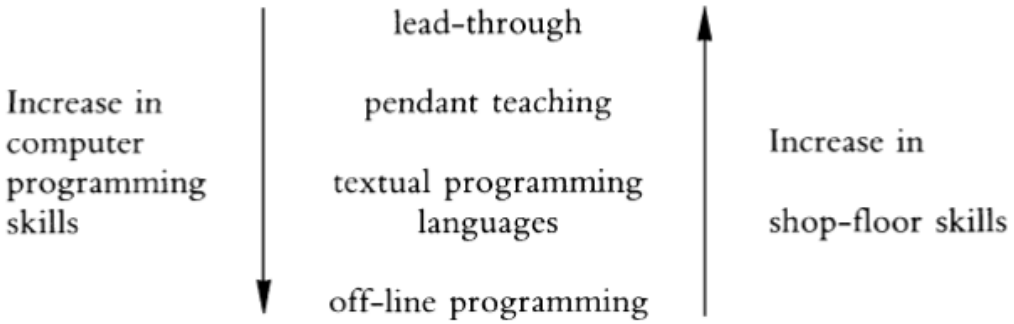

Figure 4. *Visualization of computer programming skills and shop-floor skills vs. preferred programming method. (Glagowski et al. 1992, p. 29)*

interfaces and simultaneously risen average robot user computer skills, the programming method chosen by a robot user followed more or less a rule of thumb that figure 4 illustrates.

3.1 Early robot programming languages

In 1954 the patent number US2988237A, Programmed Article Transfer, was filed to the United States Patent and Trademark Office by George Devol. The patent lead to the first robotics company Unimation and the first industrial robot ever, UNIMATE, which was installed in 1959 to General Motors die casting plant in Trenton (Robotic Industries Association 2018a). Unimate was a hydraulically actuated robot (Hägele et al. 2008) and it was programmed by recording desired joint positions. When the robot was in action, its controller compares current joint positions to the desired values and ran the joint motors simultaneously towards the desired values until they were reached. There was no programming language and programming was a tedious process. During it the programmer would increment each joint with small steps towards the target position with a trajectory that was intended and record the joint values of each of these small steps to a program, which was stored on a magnetic drum memory. There was no possibility of modifying the program afterwards or any debugging features. (Devol 1961; Robotic Industries Association 2018a)

Robot programming languages have been under development ever since and in the first two decades languages were developed in an ad-hoc manner to suit the needs of a specific robot, resulting in practically one language for each robot. In most cases the lifespan of these early languages was short. By 1982 only 8 of 22 languages developed in the last 20 years were used anymore (Poole 1989, p. 250).

Robot languages are often categorized by the abstraction level of the language and how hardware specific the language is. Many authors have drafted their own categorizations

in their textbooks and research papers (Bonner & Shin. 1982, Yong & Bonney 1999, Hägele et al. 2008) and they wary depending on the publication time and the trending programming methods at the time. As an example, the following is a categorization of robot programming language abstraction, which divides the languages based on what level of control we have on the robot (Yong & Bonney 1999 citing A. Thangaraj & M. Doelfs 1991):

1. Joint level: We have control only of individual joints, which leads to individual programming of each joint necessary.
2. Manipulator level: We can move the manipulator in terms of world/cartesian coordinates, which indicates that mathematical operations such as inverse kinematics takes care of individual joint values.
3. Object level: A task is specified with movements and positions of objects in the robot's workspace. Implies existence of a world model (more on that subject later).
4. Objective level: A task for the robot to complete is specified in the most general form, for example "paint that piece red". Implies the existence of a world model and a database with knowledge on how to do the necessary tasks.

In the next chapter this thesis will take a look at a study, that offers a snapshot of the state of the art of robot programming languages in 1982.

3.2 A comparative study of robot programming languages

In 1982 Bonner and Shin reviewed 14 robot languages of which some were widely used in industry, some were already obsolete, and some were only in experimental state. The study named five classes for categorizing robot programming languages abstraction:

1. Microcomputer/hardware level
2. Point-to-Point level
3. Primitive Motion level
4. Structured Programming level
5. Task-oriented level

Most simple methods of robot programming, hardware level languages, were not really languages at all, simply systems to read sensor data from the joints and drive the motors in the joints according to some recorded value. The values were recorded in some similar manner as in the UNIMATE system discussed previously.

A little more advanced and the most widely used programming method in industry at the time was point-to-point programming. It was the most popular choice, because the technology was simple but powerful enough and it was ready to use. Example robot languages

of this class are T3 and Funky. They provided programming methods “similar to a cassette tape recorder”; play, erase, record, reverse, and fast forward. In the programming phase the robot was maneuvered around by hand or other means to the desired points and the configuration was recorded. T3 system used a joystick for this. After this, the operator could step through a recorded program and erase or add points to it. Some system provided the possibility to have simple functions in the program to be triggered from a signal for example a limit switch on the manipulator. However, branching and subroutine capabilities, and interaction with external sensors were very limited or nonexistent. There were no possibilities to handle emergency situations and no written programming language.

Languages like VAL, RCL and RPL are categorized as primitive motion level languages by Bonner and Shin. They are written in a human readable form, much like Assembly language is compared to machine language. This enables offline programming in theory, although it wasn’t probably very practical, at least not when used as the only programming method (Fuller 1991, p. 276). The characteristics of these languages that separate them from less abstract languages are the written form of the language, better branching and subroutine possibilities often with parameters, multiple manipulators control at the same time, and introduction of frame definitions. As an example of a primitive motion language let’s examine VAL. It was initially introduced in 1975 and further developed for Unimation’s PUMA robots after 1977, was capable of point-to-point motions, joint interpolated motions, cartesian motions, simple arithmetic operations, subroutines, if-else branching and while-do-loops (Shimano 1979).

```
1.       SETI N.PARTS = 0
2. 100   VPICTURE
3.       VLOCATE PART, 100
4.       APPRO CAMERA:PART:PICK.UP, 50.
5.       MOVES CAMERA:PART:PICK.UP
6.       GRASP 25.
7.       DEPART 50.
8.       APPRO PALLET, 50.
9.       MOVES PALLET
10.      OPENI
11.      DEPART 100.
12.      SHIFT PALLET BY 5.2, 25.4, 0
13.      SETI N.PARTS = N.PARTS + 1
14.      IF N.PARTS NE 10 THEN 100
15.      STOP

        Figure 2. Sample VAL Program
```

Figure 5. *An example VAL program (Shimano 1979, p. 883)*

More high-level robot programming languages are structured programming languages. These had major improvements in programmability including wider use of control structures and complex data structures such as arrays, points, vectors and frames. The manipulator control is improved by extensive use of coordinate transformations and frames, improvement in sensors, parallel processing and using predefined state variables. State

variables keep track of the current state of the manipulator during execution time, e.g. PAL has a state variable *arm* which keeps track of the end of the robot arm with respect to the world and *tol* that keeps track of the tip of the tool with respect to the arm. These improvements contribute towards better program understandability and task level programming. The MCL system had even some primitive capabilities of machine vision; it was capable of finding and identifying some objects.

Task-oriented, or task level, programming in robotics allows the programmer to focus on what should be done rather than how the manipulator should move to accomplish the results. AUTOPASS was developed by IBM (Lieberman et al. 1977) to meet these terms. It conceals sensor management and coordinate transformations from the user and it is programmed with a natural English-like language. It provided high-level commands like "*place* object1 *on* object2". Commands meant exactly what the user thought them to mean, which is something that was not true for most of the languages in that time - most of the primitive motion languages had only six characters to describe each command. In case of ambiguities, the AUTOPASS system would have checked on the user what he/she really meant. Task-oriented programming requires the control system to have extensive knowledge about its work environment and the objects in it and the manipulator's kinematic structure. It also needs a lot of sensing abilities, e.g. some way of finding and identifying the objects around the workspace. AUTOPASS was never fully implemented, but it did establish an idea about more abstract level of robot programming. (Poole 1989, pp 250, 254).

4. TASK LEVEL PROGRAMMING

Even though some high-level languages discussed in the previous chapter seem advanced, the industry standard was point-to-point level languages. Programming with them was quite slow and tedious. Production needed to be halted during reprogramming and offline programming was not feasible to use in industry, which probably lead for reprogramming to be avoided as much as possible. For these reasons, there was a serious need to develop more efficient ways to program robots.

Task level programming, also called task-oriented programming or implicit programming (Backhaus & Reinhart 2015), has been an idea and a goal for many roboticists for over 40 years (Bonner & Shin 1982; Lozano-Pérez & Brooks 1985; Backhaus & Reinhart 2015). It aims to change the established standard and to change the way robot application programs are developed. The goal is to ease the job of factory level robot users by utilizing artificial intelligence for path planning and decision making. Task level programming aims for a situation where the robot operator concentrates on what should be done with the work objects to reach a goal, not on how the robot manipulator should move to achieve the goal. If we look at the categorization of robot programming abstraction introduced in chapter 2.2.1, robot programming has previously been on the manipulator control level of abstraction. In task level programming the abstraction level is overlapping object and objective level of control in the categorization (Yong & Bonney 1999).

4.1 A simple example

Let's examine a situation illustrated in figure 6 where there is a nicely drawn robot manipulator with three blocks in its workspace, identified with the letters A, B and C. The goal is to stack block A on top of block C. The block A needs to be precisely and firmly planted on top of C but at the same time carefully without excessive force so we won't break the blocks or the manipulator. We assume we have some sort of force-feedback from the manipulators joints.

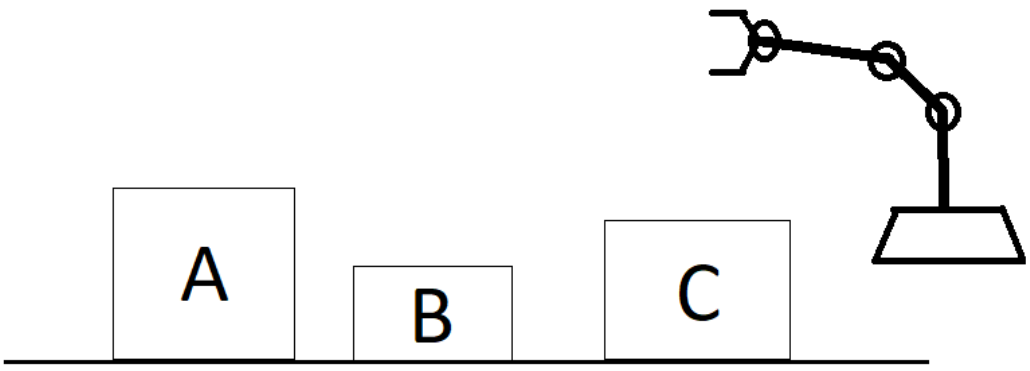

Figure 6. *The example situation illustrated*

In a traditional online programming setting using a teach pendant, the robot programmer would go through at least the following steps with a teach pendant while saving points after each step:

1. Jog the manipulator on top of A
2. Approach A with the gripper to a grasping position.
3. Grasp A.
4. Lift manipulator high enough so that A avoids collision with B.
5. Move manipulator towards C.
6. Lower gripper while controlling for the downwards force until A is firmly planted on C.
7. Ungrasp A.
8. Lift the gripper to a safe height from A to avoid collision.
9. Return home position.

In a task level programming environment this could be done simply by typing something like "*place* A *on* B *with force* X" in a UI and the environment would deduce the control code for the manipulator (Backhaus & Reinhart 2015).

In the next subchapters this thesis will be looking into the theory behind a task level system and how the previous example is possible.

4.2 Structure of a task level programming environment

This thesis will be concentrating on a proposition by Lozano-Pérez for task level programming environment's requirements. Initially the theory was defined in his 1983 paper "Robot Programming" and in his and R. Brooks' chapter "Task-level manipulator programming" of a 1985 textbook "Handbook of industrial robotics". The chapter was later refined and updated by Yong and Bonney in 1999 in the second edition of the same text book. Lozano-Pérez and Brooks (1985) defined task level programming as a process where the operator "specifies goals for the positions of objects, rather than the motions of the robot needed to achieve those goals. In particular, a task level specification is meant to be completely robot independent."

Task level programming consists of a world model, task-specifications, and motion planning. In short, a task level environment has a model of the surrounding world, it takes task-specifications in an agreed syntax as inputs and outputs manipulator-level control code that the robot executes. This kind of schema for task level programming environments can be seen in task level environments developed decades later (Shimada & Asakura 1996; Backhaus & Reinhart 2015). The next chapters will be having a look in these components.

4.2.1 World model

A central part of a task level environment is the world model. The world model is a model about the workspace, the work objects and the manipulator itself. The world model contains *geometric information*, *kinematic models* and *physical attributes* such as masses and frictions of the objects in it.

Geometric information about the objects' sizes, places and orientations are stored as CAD models. Machine vision and 3D-sensors may also be used to gain geometric information about poses and geometries of objects (Wasserman et al. 2018). Geometric information is used when planning collision-free movements for the manipulator. In the previous example of the size of block B is crucial for the path because we need to know how high A needs to be lifted to avoid collision.

The *kinematic model* of the manipulator describes the length and shape of the links and the joints' ranges, speeds, torques, directions and orientations. The kinematic models of the work objects are also essential. They contain information about linkages and constraints between the objects in the workspace, for example if we wish to turn a lever with the manipulator, we need to know how the lever moves as it is attached from its other end. Some of the work objects might be attached to each other initially and likely we will be attaching some work objects to each other during the work process that the robot is used for. These newly added constraints between the work objects need to be updated in their kinematic models as they occur. Similarly, if constraints are destroyed during operation, meaning some parts become unattached, the kinematic models need to be updated.

Physical attributes, e.g. weight of the objects determine how fast they can be moved and how much force do we need to assert to move them. Information about frictions in the various surfaces in the workspace might also be necessary for intelligent object manipulation.

In addition to a precise world model, sensing the actual work environment is crucial in keeping track of the situation and it allows us to use the manipulator more diversely. Machine vision provides verification about the position of the work objects. Touch sensing in the hand of the manipulator tells us when contact with another surface is made, since the world model might not be totally accurate. Force sensors in the manipulator enable precise pressures with the end-effector against other surfaces and torque measurements that are necessary in assembly tasks.

4.2.2 The task specification

Tasks can be defined as sequences of states of the world model. Figure 7 illustrates an assembly task defined by five model states. Proceeding from left to right the states of the shaft explain the assembly procedure a step at a time.

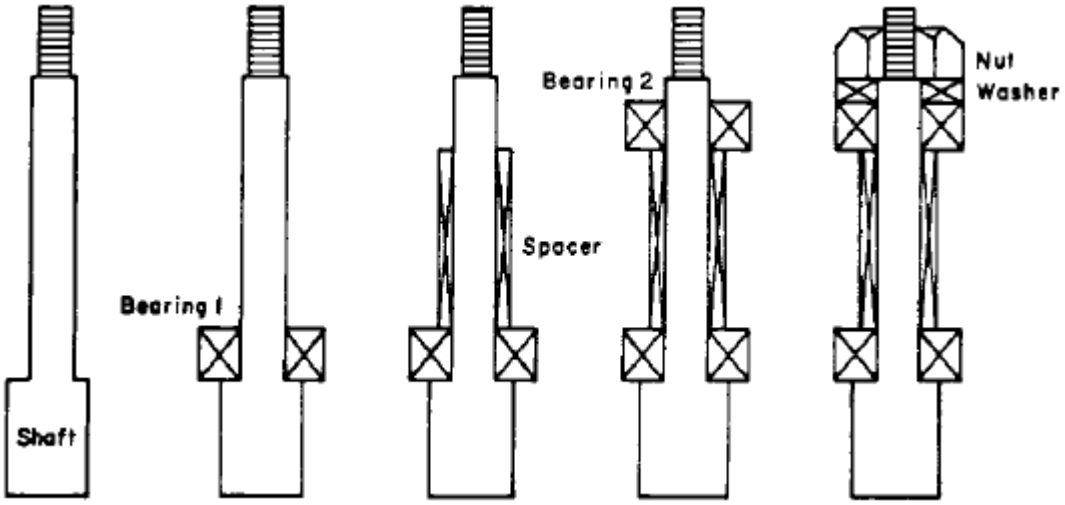

Figure 7. *An example state description of an assembly task (Lozano-Pérez 1983, p. 836)*

A task specification consists from two to any number of model states. In a single model state each of the positions of all objects in the environment is described. Lozano-Pérez proposes two relevant ways to describe these model states:

1. Using CAD models. The sequence of states described in figure 7 could very well be in CAD format. With information about the sizes, masses and locations of every object, a task level system could synthesize the necessary movements for the manipulator.

2. Using symbolic expressions to describe the spatial relationships between the objects in the work environment. For example, let's describe the position of block 1

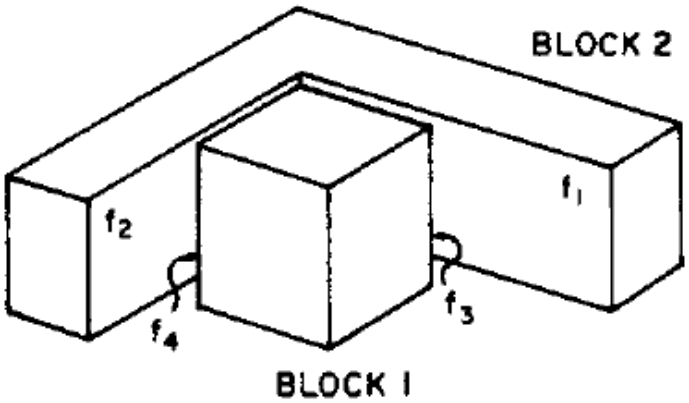

Figure 8. *Configuration of block 1 and block 2 (Lozano-Pérez 1983, p. 826)*

in relation to block 2 illustrated in figure 8 using symbolic expressions: (f3 against f 1) AND (f4 against f2), where f1, f2, f3 and f4 are faces of the blocks.

The underdeveloped state of CAD/CAM systems was a major obstacle in the way of task level programming in the 1980's (Lozano-Pérez 1983). However, CAD-technology has matured from those times and it is not a problem anymore to model even the most complicated workspaces and objects with good precision.

The task specification might need some additional information in addition to the geometric data. Let's take a situation where we are tightening a bolt as an example: The CAD model states or the symbolic state expressions describing the states doesn't necessarily tell us how much force need to be used to tighten the bolt. This needs to be specified elsewhere. (Lozano-Pérez 1983)

Previously task specifications were described as sequences of state models. Alternative way of describing the task could be by a sequence of operations. The task specification of the assembly from figure 7 could be described as following also (Lozano-Pérez 1983):

1. PLACE BEARING1 SO (SHAFT FITS BEARING1.HOLE) AND (BEARING1.BOTTOM AGAINST SHAFT.LIP)
2. PLACE SPACER SO (SHAFT FITS SPACER.HOLE) AND (SPACER.BOTTOM AGAINST BEARING1.TOP)
3. PLACE BEARING SO (SHAFT FITS BEARING2.HOLE) AND (BEARING2.BOTTOM AGAINST SPACER.TOP)
4. PLACE WASHER SO (SHAFT FITS WASHER.HOLE) AND (WASHER.BOTTOM AGAINST BEARING2.TOP)
5. SCREW-IN NUT ON SHAFT TO (TORQUE = t0)

4.2.3 Interpretation of the task specification

Before the manipulator movements can be planned, the inputted task specifications need to be broken down in to smaller parts, sub-tasks, which represent single manipulator movements implicitly on a reference level and these sub-tasks need to be arranged to a correct chronological order to complete the task.

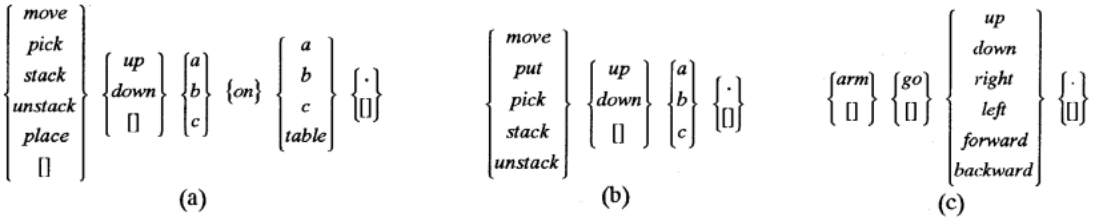

Figure 9. *Three templates a, b and c for a sentence that specifies a task. (Shimada & Asakura 1996, p. 1229)*

A very simple way to deconstruct a pick-and-place task specification is illustrated in figure 9. A sentence is divided in to words, from which sub-tasks such as *move* or *pick* are recognized. These commands are possibly followed by parameters specifying the command. The identified sub-tasks are then arranged as a stack for execution. (Shimada & Asakura 1996)

Usually a hierarchical way of constructing a task description is more useful (Nagai et al. 2007; Knoop et al. 2008; Backhaus & Reinhart 2015). This way allows complicated, scalable tasks, which are logically reduced to indivisible sub-tasks. Figure 10 represents a pick-and-place task as a hierarchical tree structure.

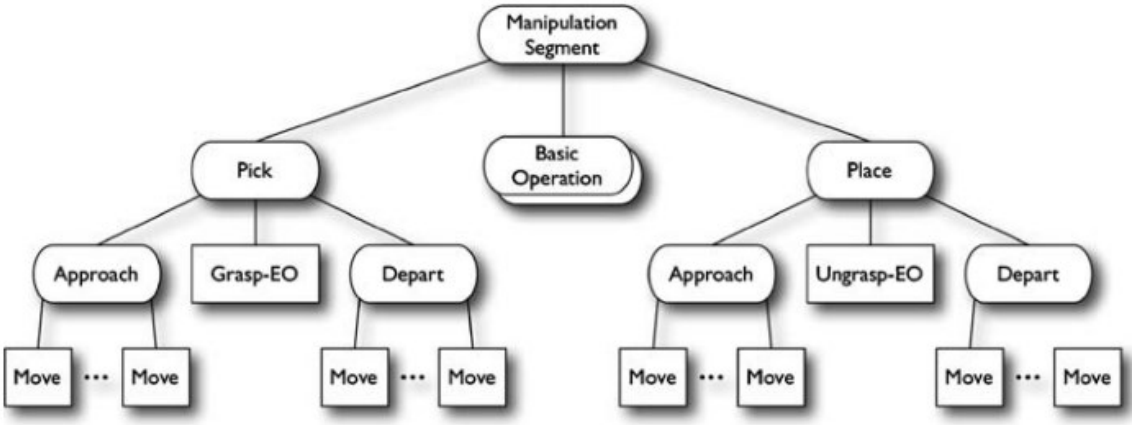

Figure 10. *Singular operations that make up a manipulation task (Knoop et al. 2008, p. 347)*

A taxonomy for assembly process is presented in Martinéz Lastra's dissertation "Reference Mechatronic Architercture for Actor-based Assembly systems" from 2004. He focuses on the manufacturing process of assembly, but similar logic can be applied to other manufacturing processes as well. In figure 11 the deconstruction of an assembly process to single operations is illustrated.

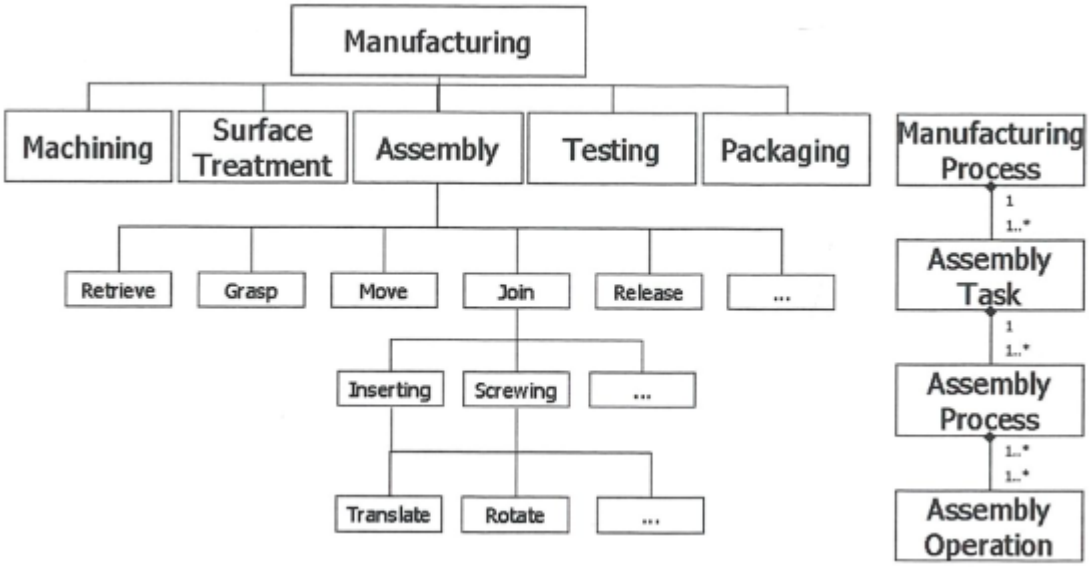

Figure 11. *Deconstruction of an assembly process (Martinéz Lastra 2004, p. 33)*

A manufacturing process encapsulates all tasks that are involved in it. A task, assembly task in this case, is a task level description of a job. A task is composed from processes and a process from operations. Single operations might be for example translation, rotation, force application or torque application, which change only one aspect of the work object at hand.

The task specification can be interpreted to robot code with a task planner. As an example, Backhaus and Reinhart created a device independent task planning module for assembly systems in their 2015 study. The planning module utilizes a set of knowledge sources while planning the task. The input is a task description as an Automation Markup Language-file (AML), which is an XML-based data format. The output is vendor specific robot code to complete the task. The task planning is an iterative process, adding more information about the task on each level. The information of each level is fetched from different knowledge sources, which are not defined in full detail in the paper. A knowledge source needs to identify what kind of information it provides, specify when the knowledge source is triggered and it of course needs to output some sort of an action for the situation, for example a piece of code or an algorithm. The knowledge source might also be the user in some uncertain cases. Next, some terms that are used later are explained similar how Backhaus and Reinhardt define them.

- Process is an abstract description of an operation. A process always has relations to one or more products and is solution-neutral, so they can be executed by one or more skills and therefore resources.
- Product is an object that is handled in an assembly process.
- Resource is a machine or component which transports or handles sub-products or products. A resource might have controllable functions, which are represented as skills.

- Skill is an ability to perform an operation. A skill represents an abstract and vendor independent function that are mapped to device (resource) functions depending on the physical setup.

The iterative task planning process consists of the following five level hierarchy:

- Task level (1): Only the joining (primary) processes and the relation to the product and product parts are described.
- Assembly sequence level (2): In addition to the information in the task level the sequence of the joining processes is defined.
- Process level (3): All relevant primary processes are assigned to skills and necessary secondary processes as well as associated skills are identified based on the assembly sequence level.
- Skill level (4): The sequence of all elemental operations is planned based on the skills.
- Code level (5): The code level represents the vendor specific code for all controls in the assembly system.

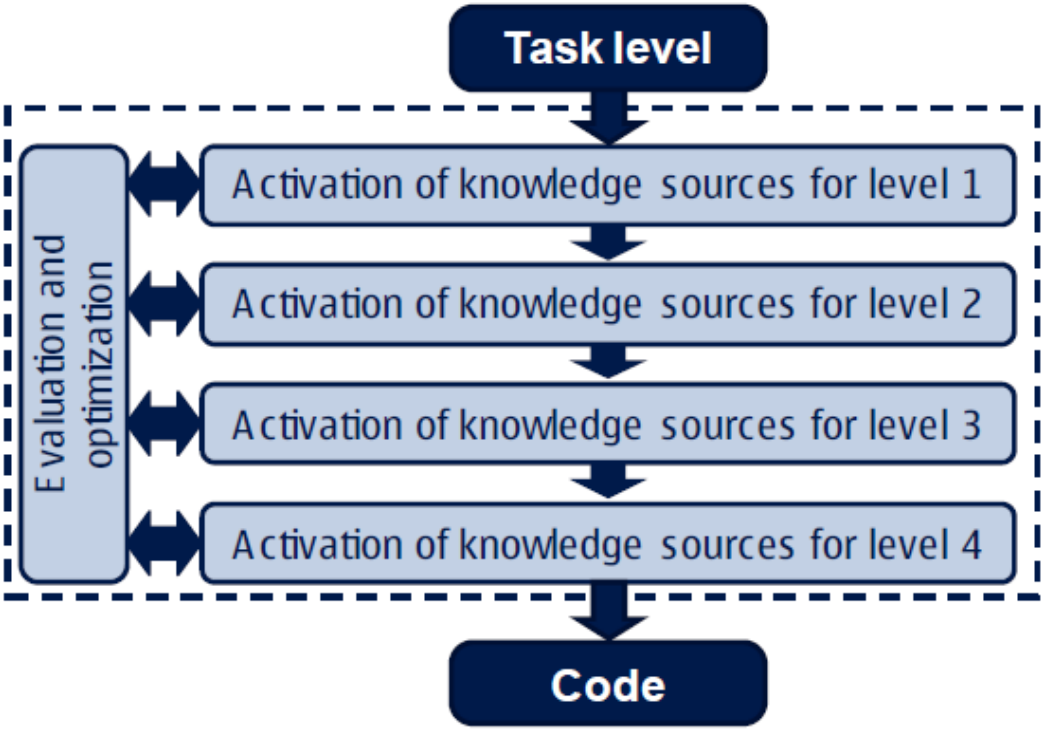

Figure 12. *Task planning process sequence. (Backhaus & Reinhart 2015, p. 548)*

Each time new information is presented to the task description, the robot program of the current situation is revised and optimized, bad path decisions are discarded on basis of relevant parameters such as cycle time or energy consumption. The main sequence is presented in figure 12.

4.2.4 Motion planning

With the implicit sub-tasks, object locations and geometries, as well as manipulator kinematic structure, the movements can be planned explicitly.

A task often involves various types of movements during its execution. A manipulator should move at different velocities and accelerations when the end effector is moving large distances and there is no immediate danger of collision compared to when the end-effector is in immediate proximity or in contact with another surface, like in welding or grinding a work piece. Motion planning can be divided into gross motion planning, grasping planning and fine motion planning (Lozano-Pérez & Brooks 1985). There are several path planning and collision avoidance algorithms, which will not be further discussed here.

Gross motions are movements of the manipulator where the only constraint is collision avoidance with other objects. The purpose of gross motion planning is to look for the most efficient path from the manipulator's initial orientation to the goal orientation while avoiding collisions.

Grasping planning searches for the optimal way to grasp a work object with a gripper. The process plans the grasping orientation according to three principal rules (Yong & Bonney 1999):

1. Safety. The robot must be safe at the initial and final grasp positions.
2. Reachability. The robot must reach the initial grasping position and with the object in hand, find a collision free path to the final position.
3. Stability. The grasp should be stable and withstand all forces directed against the object during transfer movement and possible parts-joining operations at the final position.

Fine motion planning is needed when the manipulator is approaching a surface of an object and the accuracy of the world model is not sufficient to rule out the possibility of a collision. Another situation when fine motion planning is needed is compliant motion, which is when the manipulator is in contact with a surface. Especially in compliant motion the path planning is special, because we now have a new constraint in the equation; to maintain the contact with the surface.

4.3 Development of task level programming

Next, a couple of example task level robot systems that have been developed in the past will be reviewed briefly. The current state of the industry will be reviewed in the next chapter.

One of the first early attempts at a task level programming language was the previously mentioned task-oriented language AUTOPASS project by IBM in the mid 1970's. As mentioned, it never was fully implemented but it still laid some ground work for later attempts (Poole 1989, pp. 250, 254).

In 1987 and 1989 Lozano-Pérez et al. from MIT published research papers on their task level robot system Handey. It was the first or at least one of the first task level systems that successfully completed complex tasks. Handey was a task level robot system for pick-and-place operations for assembly tasks. Handey required a world model including geometric and kinematic models of the part to be manipulated, the objects in the work-space and the manipulator. With the models and a specified desired position of the work object it could pick and place work objects from and to obstructed areas. Handey plans a grasping motion to the object, plans a collision free movement to the end position and places the object there. If the work object could be grasped at the initial position, but the same grasp was not feasible for the desired position, Handey added a re-grasping sequence in its task plan, where the work object was grasped at a safe location in a way that the work object could be placed to the desired end-position. The 1989 research was conducted with Unimation's PUMA robot, but Handey-system worked with any robot when a manipulator specific model is provided. (Lozano-Pérez et al. 1987; Lozano-Pérez et al. 1989)

In 1997 Kawasaki Heavy Industries and Matsuhita Electric Industrial developed a prototype task level welding robot system for a Japan Robot Association's program called "Robot programming simplification project". The UI for the system runs on a PC, which is connected via LAN to another, more powerful computer running a 3D-CAD system that hosts the world model. The PC is also connected to the robot controller with a serial connection. The user would create a task-specification on the PC in a responsive and intuitive graphical UI selecting the welding seams on a CAD-model and setting all the necessary welding parameters. After that the program is simulated and the user can either modify it or confirm it for motion planning, after which it is finally sent to the robot controller. (Arai et al. 1997) The testing period of the system proved that teaching times can be reduced down to one-sixth compared to some other widely used robot programing system of the time (Y. Yong & Bonney 1999).

In 2002 Rosell et al. developed a task level robot system, for polishing and grinding tasks. It takes models of the workcell and work objects as input and outputs an execution file for the robot and a 3D-simulation file, which the user can use to verify the created program. It followed a CAD based approach, where the user specifies the polishing or grinding task by selecting curves from the surface of the work object's CAD model to be grinded or polished. Different pressures exerted for grinding or polishing can also be selected. The tool plans manipulator movements in a time-optimum and collision free way for the task specification. According to the authors, the approach could be easily extended to other tasks such as cutting or material dispensing. (Rosell et al. 2002)

4.4 State of the art

By default, industrial robots from major suppliers are programmed with a jogging/lead-through approach or an offline programming approach. An offline programming environment, where the user specifies how the robot should perform a task by jogging a 3D-model of the robot is not quite task-level programming, since the user still does the overall planning (Hägele et al. 2008). However, during 2010's many major industrial robot manufacturers such as ABB, Fanuc and Yaskawa have included some level of automatic path planning in their environments, where the environment automatically creates end-effector paths along CAD-model edges and/or surfaces. Let's take Fanuc's offline programming environment RoboGuide as an example. Fanuc's RoboGuide in its standard form without additional modules enables creation of workcell layouts, offline programming with code input or a virtual teach pendant and visualizing robot programs in offline environment. Fanuc also offers optional software packages to RoboGuide for application specific improvements in RoboGuide performance, usability and automated program creation from workpiece models. For example, WeldPRO-module automatically creates a program from shape data of a workpiece. The user can select an edge in the workpiece model and WeldPRO creates an arc welding program for the robot arm, where tool orientation is kept at a designated angle relative to the welding path. Other modules for RobotGuide include ChamferingPRO for chamfering applications and PaintPRO for painting applications, from which both of them also provide automated path planning capabilities from work piece CAD-data (Fanuc 2017).

Next, some commercial products and studies that represent the state-of-the-art in ways of programming industrial robots with a task-oriented approach in mind will be presented. The user of these systems can focus on what should be done to complete a task instead of how the robot should work to achieve a task. While searching for commercial task level robot programming systems I found a couple of interesting application specific systems, which are not from major robot suppliers: RinasWeld by a Dutch company Kranendonk and RobotMaster v7 by Hypertherm Inc.

The latest version of RobotMaster, RobotMaster v7 was launched in 2018. It is an offline robot programming system for different industrial applications, such as welding and contouring tasks the latter including trimming, cutting and deburring. The future releases of the software will include modules for assembly, surfacing, 3D-milling and additive manufacturing. It is advertised to be a very intuitive, easy-to-use, task-based system, which can be operated by the process expert without programming or CAD expertise. It supports any number of robots, tools, configurations, tasks or processes and manages control of external axes, such as linear rails, workpiece positioners and rotary tables. The system offers a bi-directional integration with CAD-programs that allows Robotmaster to assess the part design and CAD-programs to receive data from Robotmaster. Future releases will include criteria-based optimization, which allows the user to choose most desirable criteria used when the system chooses robot paths from all available paths. These criteria can

be e.g. to minimize motion of a specific joint, cycle time or robot rigidity. (Robotic Industries Association 2018b; RobotMaster 2018a; CompositeWorld 2018)

In Robotmaster the user interacts with a graphical user interface unique and modifiable to suit the needs of each application. The simulated workcell is configured using CAD-models of manipulators, work objects, external axes and other objects included in the real workcell. When creating robot tasks, the user selects surfaces, edges or other elements of the working objects' CAD-models and depending on the used tool and task the software automatically creates a path for the manipulator and controls for the used tool. Robotworks creates trajectories and deals with reach issues, joint limits, singularities and collisions automatically. After a task has been created the user verifies it feasible and the program outputs a robot control program. Robotmaster offers turn-key solutions for most major robot brands such as ABB, Fanuc, Kuka and Yaskawa (RobotMaster 2018b).

Another company, Kranendonk supplies complete production lines for robotic welding, cutting and assembly. RinasWeld is their offline programming environment for programming of steel beam assembly and welding tasks and it is developed especially for small batch production and continuously changing designs keeping the programming time at minimum. Its goal is to automate the robot and welding programming as far as possible and require as little as possible input from the user. In RinasWeld the simulated workcell is statically hardcoded into the environment for each customer and workcell. The system supports multiple robots in the workcell that can either be welding or manipulating the work objects. (Larkin et al. 2011, Kranendonk 2015a)

When creating a new program in RinasWeld, the user first imports a CAD-assembly of the desired steel beam assembly to the programming environment. Next, the program automatically deduces where the welding seams should be between the components in the assembly. After this, the user can edit the welds if necessary and confirm the design. After user verification, the program creates a collision-free robot control program in a manufacturer specific language depending on the produced welding design and workcell configuration completely autonomously without any code input needed from the user. The created program controls the welding robots and part manipulating robots as well. RinasWeld also finds all the necessary welding parameters necessary for the torch control from a database of approved weld parameters. The database can be modified to satisfy the user's needs and used materials. According to a 2011 study by Larkin et al., the database for welding parameters can be easily expanded easily with welding parameters for different types of welds and materials. Larkin also says that RinasWeld tolerates errors between real-world and simulated environments very well with the use of touch sense calibration. (Larkin et al. 2011; Kranendonk 2015a; Kranendonk 2015b)

Finding intuitive ways of human-robot interaction (HRI) is closely related to robot programming on a task level, since a natural way of describing the task to the robotic system reduces needed programming skills. Recently there has been rising interest regarding

more interactive HRI in literature. The emerging methods for achieving these easier interfaces have been multi-modal interfaces (MMI), programming by demonstration (PbD), virtual reality (VR) and augmented reality (AR) (Fang et al. 2014). Next, some studies with the topic of programming robots with these methods are reviewed.

AR can help a robot programmer to interact intuitively with the workspace objects and manipulator trajectories by displaying relevant virtual information in the actual working environment. Wasserman et al. utilized AR and machine vision in their 2018 study regarding intuitive robot programming. They developed an environment perception module, an augmented reality simulation module and an automated program verification module and integrated them into a pre-existing cloud-based robot control system, which includes a task-oriented programming interface. The modules enable object recognition in a 3D-space and verification of collision free manipulator movements. The workcell is monitored with a depth-image camera. All objects that differ from a set base plane, marked with a 2D-marker, are recognized and a bounding box is set around them corresponding to the objects size and orientation. This is visible in figure 13.

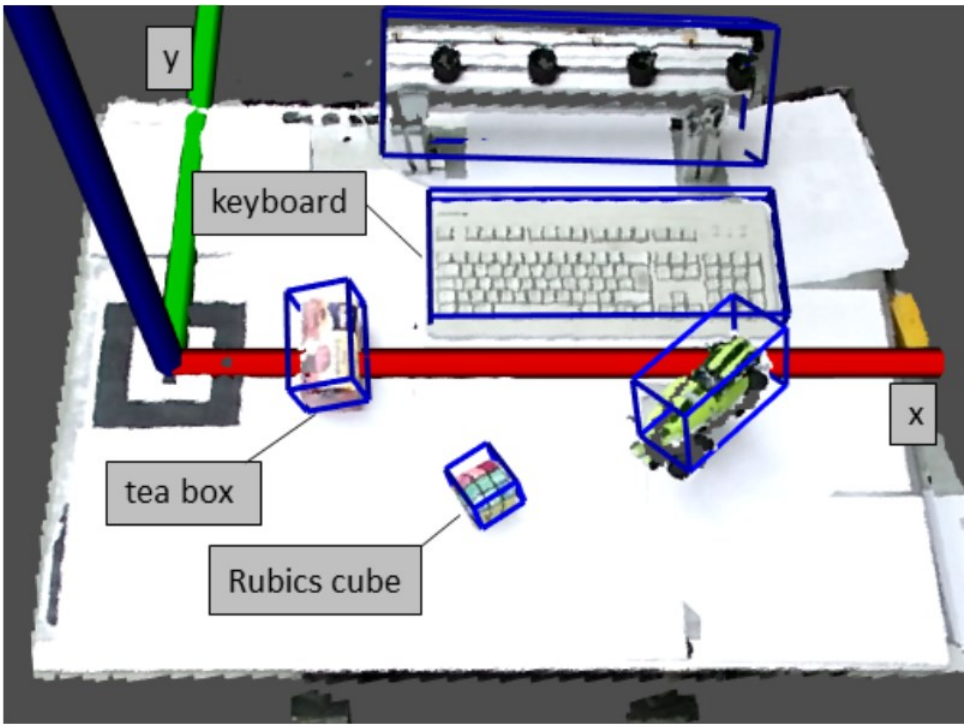

Figure 13. *Objects in the workspace and their surrounding boxes (Wassermann et al. 2018 p. 164)*

With the help of these 3D-boxes collision-free manipulator path verification is possible. In the study, the collision avoidance is checked with a simulated robot manipulator before sending the program to a real robot.

In figure 14 below, the robot's workspace is filled with arbitrary collision boxes and the ones which the simulated robot collides with are colored red.

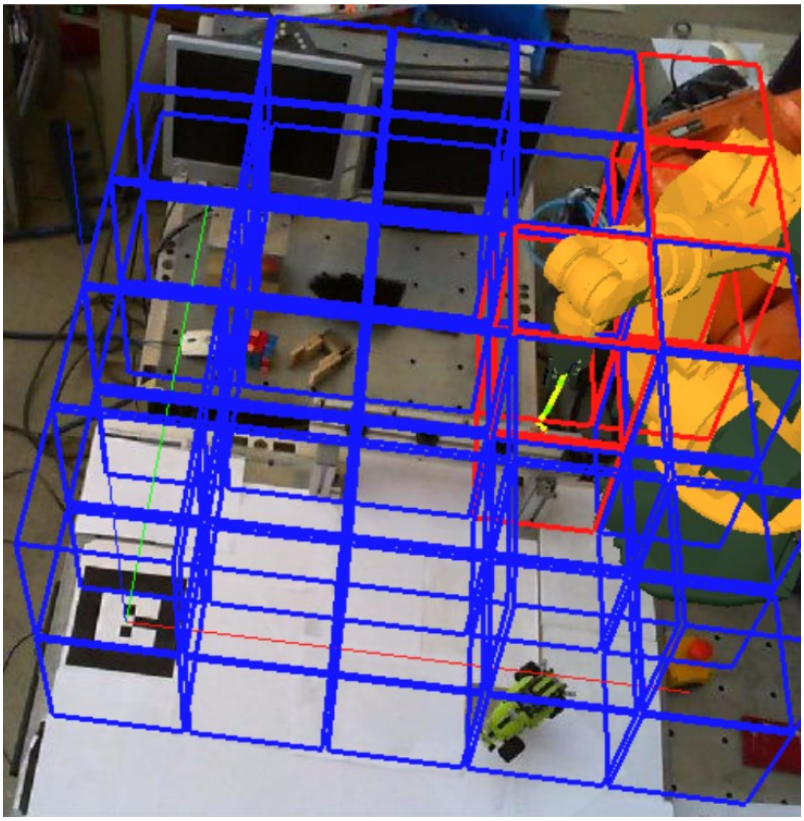

Figure 14. *Workspace filled with bounding boxes, the ones that cause collision with our manipulator are colored red. (Wassermann et al. 2018 p. 165)*

MMI in robotics means using natural human interfaces, which can be for example speech or gestures, alongside with or instead of conventional programming methods, such as the teach pendant or an offline programming environment. For an example, a 2010 study by Neto et al. replaced the conventional teaching pendant with motion and speech commands. The study proposes a multimodal robotic system, which uses a Wii remote and a Bluetooth headset as the human-machine interface (HMI). From the Wii remote the system collects hand gestures and translates them into end-effector movements. Voice commands are collected via the headset and the user can for example build the robot code in general language or stop the motors of the manipulator using speech.

The system was taught to recognize each user's hand gestures with an artificial neural network. The number of learning examples ranged from 20-70 and the recognition rates of tested gestures ranged from 70% to 100% depending on training examples count, tested gesture, and test person. For the intuitive system to be safe to use for unexperienced users some force-feedback is necessary. The system gets force and torque information from all three axes from a force/torque sensor positioned in the end-effector. The system alerts the user of excessive force against a surface by vibrating the Wii remote. The alert is sent if

the force exceeds a set value. If the force continues to grow and exceeds a limit value, the robot immediately stops all movement. Force control is also utilized for collision avoidance in the system.

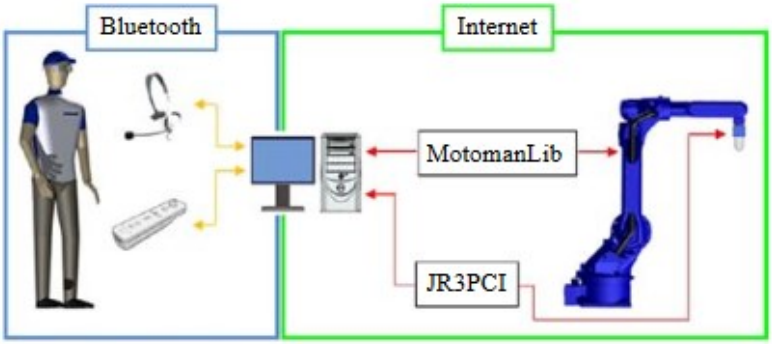

Figure 15. *Illustration of the HRI. JR3PCI is an ActiveX component used to communicate with the F/T sensor (Neto et al. p. 139)*

The system by Neto et al. was tested with pick-and-place operations and writing text with a pen to a paper. The time spent on teaching the writing task to the system was compared to teaching the same task with a manually guided lead through robot with a similar F/T sensor. The lead-through robot turned out to be 30% faster to use at the time, but the authors were confident that their system could be more or equally practical as the manually guided system with further development on the hand gesture and posture recognition.

PbD is a robot programming approach, where the user, or the teacher, shows how a task is done and the robot learns it by observing the teacher. The teaching data might be start and end situations of the workspace e.g. as images (Ekvall & Kragic 2008) or the actual human movements of the teacher (Aleotti & Caselli 2010; Ficuciello et al. 2014). Key issues in PdB are the imperfections in the teacher's demonstrations; the teacher might do unnecessary steps when he/she demonstrates a task and there might be a lot of noise in the teaching examples, meaning different variations of the task (Fang et al. 2014). Another issue is that the initial situation of the task might be different of what has been taught and the task is impossible to execute exactly as taught. Answer to this is to have the PbD system segment full tasks in to sub-tasks and recognize their relationships (Ekvall & Kragic 2008).

Some PbD applications have been proposed in studies. Aleotti and Caselli proposed a PbD way of programming robotic grasps in a VR environment. In their paper, they used a virtual reality glove and recorded human hand grasping items in a VR environment. The human grasps were then mapped to robot gripper grasps and the robot grasp's quality was assessed with a quality measuring system that was also proposed in the same paper.

Another PbD related study was done by Ekvall & Kragic, where they presented a PbD task learning system in to a pick and place scenario. The system was tested with in a

virtual simulation and using a real robot. In the scenario a robotic manipulator was supposed to arrange three work objects in the workspace as taught. The initial state of the workspace was slightly different in the beginning of each task execution and in real world tests the initial state was estimated using a camera.

Before the task execution the system was trained for the task with three different PbD learning techniques in three different test situations; imitation learning, learning in dialogue with the teacher and learning with multiple observations. In real world tests the training input was pictures of desired end situations. In imitation learning the system is given only the end result and it tries to replicate the same result. In learning in dialogue with the teacher the task might be shown only once, but the teacher can tell the system additional constraints for each step. For example, the robot can be told to place a certain object before any other object. In learning with multiple observations, the task is taught multiple times, which makes it possible to form a more general model of the task. This is done by extracting meaningful constraints from the teaching examples e.g. the sequence certain sub-tasks in a larger set of sub-tasks, or by eliminating unnecessary sub-tasks that do not occur in all of the teaching examples. In Ekvall and Kragic's paper the multiple examples learning technique was tested by having underlying constraints, such as always place a certain object first, or that some object needs to be positioned in relation to another object instead of the world frame.

A more recent PbD study from 2017 was conducted by Papadopoulos et al. They presented an advanced HRI interface that allows non-expert users to teach assembly tasks for a collaborative robot simply by showing to a camera how the assembly is done. In the study an ABB irb-14000 (YuMi) collaborative robot was used. The interface uses a RGBD-camera and a 3D-simulation of the workspace, which the user can follow the teaching of the assembly.

Using the interface works as following. First, the human creates a task with a name, specified assembly parts and an assembly type, e.g. insertion. The specified parts appear in the 3D-simulation. Then, the parts are detected in the actual work environment with the RGBD camera, and the system shows the detected poses of the parts in the 3D-simulation. After this, the teacher shows how the assembly is done in front of the RGBD-camera, which records a video of the assembly. From the video, the system extracts main frames, meaning frames that demonstrate movements of the parts. These frames can be edited by the user if they are not satisfying. Additional information can also be added to each frame if needed, for example information about two parts aligning with each other. After this the assembly (without robot movement) is displayed to the user. Once the assembly is taught and verified by the user, the system designs 3D-models for optimal gripper fingers to use for the task, which are printed and attached to the robot by the user. Next, the system creates manipulator paths and gripping positions for the assembly in the simulation, which the user confirms if feasible after which the robot program can be transferred to a real robot. Papadopoulos et al. conducted an experiment with 13 inexperienced users

to evaluate the system's usability on a five-point Likert scale. All of them rated the system very positively and felt like they could use the system without any assistance. (Papadopoulos et al. 2017)

5. CONCLUSIONS

Task level robot programming is the highest level of abstraction in robot programming and a way to program a robot at a task level. The user does not teach the robot how to do, but instead instructs it by what to do. A task level programming environment normally includes a model of the work environment including work objects and machinery, some sort of a set template that the user uses when he/she inputs a task description and a way to plan the robot program from that task description. The goal of task level programming is to relieve the robot operator from the necessity of knowing programming so that the operator can be a process expert instead of a programmer. This is desirable since a process expert is much more knowledgeable of the actual job, while a programmer without any process knowledge just works according to the given specification, which might be incomplete or have mistakes in them that were not apparent in the time of making the specification. Personnel with expertise in both the process and programming knowledge might be difficult to find and hire.

Robot programming in the past was tedious and reprogramming required lots of time and a robot programming expert on the worksite. This created a need for faster, better ways of communicating with the robot. HMIs have become a lot easier and more efficient to use over the years, but there still is a lot of work to do for an unskilled person to be able to program a robot (P. Neto et al. 2010). Traditional online and offline approaches require a good technical understanding of robots and programming and are economically feasible only with large batch sizes (P. Neto et al. 2010; Wassermann et al. 2018). SMEs with frequently changing products do not necessarily have qualified personnel for the robot programming jobs (Wassermann et al. 2018) and their product batch sizes are often times small. Higher level of abstraction in robot programming interfaces is needed to enable SMEs to really benefit from production efficiency gained from using robots. Previously mentioned points serve as motivation for developing task level and other high-level robot programming methods.

Working concepts of task level environments, that actually make the robot programming work more efficient have existed for 20 years now (Arai et al. 1997), but still task level programming environments are not yet common in industrial scale (Hägele et al. 2016). The transition to industry is very much needed, but it is held back by the disadvantages of task level programming. It takes a lot of effort to adapt a task level system to the components and processes of a production system (Backhaus & Reinhart 2013). However, it seems like the situation is changing judging by the latest literature proposing advanced robot programming systems and recently emerged commercial task level systems that were covered in chapter 4.4. We might see task level robot programming environments in wider use in the near future.

To make robot teaching even more natural to non-expert people, there has been a lot of task level robot programming research concentrating on natural HRI, such as multimodal interaction, augmented-reality, virtual reality and learning by demonstration (Fang et al. 2014). Even though this thesis has discussed task level programming in the scope of industrial robotics, task level programming and HRI research has coincidentally advanced robot programming over all to a higher level of abstraction. The knowledge that has been acquired can be utilized outside the industrial environment in a likely situation in the future, where normal people interact with service and professional robots on a regular basis in their jobs and free time.

REFERENCES

ABB (2017), Getting started tutorials for RobotStudio, Available at: (accessed 28.12.17) http://new.abb.com/products/robotics/robotstudio/how-to-use-it/getting-started

Ahrens G. (2001) Software companies respond to needed improvements in simulation, offline programming, Robotics World; Atlanta Vol. 19, Issue 2, pp. 26–28.

Aleotti J., Caselli S. (2010) Interactive teaching of task-oriented robot grasps, Robotics and Autonomous Systems, Volume 58, Issue 5, pp. 539–550

Arai T., Miki O., Kawaguchi H., Itoko T., Yago H., Hirayama M. (1997) A Task-level Visual Robot Teaching System, Proceedings of the 1997 EEE International Symposium on Assembly and Task Planning Marina del Rey, CA, pp. 31–35

Backhaus J., Reinhart G. (2013) Effcient Application of Task-oriented Programming for Assembly Systems, 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 750–755.

Backhaus J., Reinhart G. (2015) Adaptive and device independent planning module for task-oriented programming of assembly systems, 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering, pp. 544–549.

Bonner S., Shin K. (1982) A comparative study of robot languages, Computer, Vol. 15, Issue: 12, pp. 82–96.

Brunete A., Mateo C., Gambao E., Hernando M., Koskinen J., Ahola J., Seppälä T. Heikkilä T. (2017) User-friendly task level programming based on an online walk-through teaching approach, Industrial Robot: An International Journal, Vol. 43 Issue: 2, pp. 153–163.

CompositeWorld (2018), Robotmaster V7 offers CAD/CAM integration, available at: (accessed 08.10.18) https://www.compositesworld.com/products/robotmaster-v7-offers-cadcam-integration.

Deisenroth M., Krishnan K., On-line programming, in Nof S. (1999) Handbook of industrial robotics: Second edition, John Wiley & Sons, pp. 337–352.

Deisenroth M., Robot teaching, in: Nof S. (1985) Handbook of industrial robotics, John Wiley & Sons, pp. 352–365.

Devol G. (1961) Programmed Article Transfer Patent US2988237A.

Ekvall S., Kragic D. (2008) Robot Learning from Demonstration: A Task-level Planning Approach, International Journal of Advanced Robotic Systems, 11/2008, Vol. 5, Issue 3, pp. 223–234.

Fang H. C., Ong S. K., Nee A. Y. C. (2014) A novel augmented reality-based interface for robot path planning, International Journal on Interactive Design and Manufacturing (IJIDeM), vol. 8, no. 1, pp. 33–42.

Fanuc (2017), Fanuc RoboGuide brochure, Available at: (accessed 08.10.18) https://www.fanuc.co.jp/en/product/catalog/pdf/robot/ROBOGUIDE(E)-10.pdf

Ficuciello F., Romano A., Lippiello V., Villani L., Siciliano B., Human Motion Mapping to a Robot Arm with Redundancy Resolution, In: Lenarčič J., Khatib O. (eds) Advances in Robot Kinematics (2014), Springer, pp. 193–201

Glagowski T., Pedram H., Shamash Y., Human factors in robot teach programming, in: Karwowski W., Rahimi M. (1992) Human-Robot Interaction, CRC Press, pp. 16–47.

Hägele M., Nilsson K., Pires J.N., Bischoff R., Industrial Robotics, in: Siciliano B., Khatib O. (2016) Handbook of Robotics 2nd Edition, Springer, pp. 1385–1421.

Hägele M., Nilsson K., Pires J.N., Industrial Robotics, in: Siciliano B., Khatib O. (2008) Handbook of Robotics, Springer, pp. 963–985.

International Federation of Robotics (2017) Executive Summary World Robotics 2017 Service Robots. Available at: (accessed 23.02.18) https://ifr.org/downloads/press/Executive_Summary_WR_Service_Robots_2017_1.pdf.

International Federation of Robotics (2017) Executive Summary World Robotics 2017 Industrial Robots. Available at: (accessed 23.02.18) https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf

International Federation of Robotics (2018) Industrial robot sales increase worldwide by 31 percent, press release. Available at: (accessed 01.08.18) https://ifr.org/downloads/press2018/2018-06-20_Press_Release_IFR_2017_growth_EN_rev.pdf.

J. Fuller (1991) Robotics: introduction, programming, and projects, Macmillan, 465 p.

Knoop S., Pardowitz M., Dillman R. (2008) From Abstract Task Knowledge to Executable Robot Programs, Journal of Intelligent & Robotic Systems; Dordrecht Vol. 52, Issue 3-4, pp. 343–362.

Kranendonk (2015a), RinasWeld - 'Hands-off robot programming', available at: (accessed 08.10.18) https://www.kranendonk.com/software/rinasweld.

Kranendonk (2015b), RinasWeld software in university research, available at: (accessed 08.10.18) https://www.kranendonk.com/blog/case-university-wollongong-australia.

Larkin N., Milojevic A., Zengxi P., Polden J. Norrish J. (2011) Offline programming for short batch robotic welding, 16th Joining of Materials (JOM) conference 2012 pp. 1–6.

Lozano-Pérez T. (1983) Robot programming, Proceedings of the IEEE vol. 71, no. 7, pp. 821–841.

Lozano-Pérez T., Brooks R., Task-level manipulator programming, in: Nof S. (1985) Handbook of industrial robotics, John Wiley & Sons, pp. 404–418.

Lozano-Pérez T., Jones J., Mazer E., O'Donnell P. (1989) Task-Level Planning of Pick-and-Place Robot Motions, Proceedings. 1987 IEEE International Conference on Robotics and Automation, pp. 21–29.

Lozano-Pérez T., Jones J., Mazer E., O'Donnell P., Grimson W., Tournassoud P., Lanusse A. (1987) Handey: A Robot System that Recognizes, Plans, and Manipulates, Computer (Volume: 22, Issue: 3), pp. 843–849.

Martinéz Lastra J. L. (2004), Reference Mechatronic Architecture for Actor-based Assembly Systems, dissertation, Tampere University of Technology, Publication 484, 112 p.

Motion Control Robotics (2017), Save Money by Removing the Teach Pendant. Available at: (accessed 20.04.18) http://motioncontrolsrobotics.com/save-money-by-removing-the-teach-pendant/.

Nagai T., Aramaki S., Nagasawa I. (2007) Representation and Programming for a Robotic Assembly Task Using an Assembly Structure, 7th IEEE International Conference on Computer and Information Technology. Fukushima, Japan, pp. 909–914.

Neto P., Pires J.N., Moriera A.P. (2010) High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition, Industrial Robot: An International Journal, Vol. 37 Issue: 2, pp. 137–147.

New job for the PC: Calibrating robots programmed off-line (1995), The Industrial Robot; Bedford Vol. 22, Issue 1

Papadopoulos C., Mariolis I., Topalidou-Kyniazopoulou A., Piperagkas G., Ioannidis D., and Tzovaras D. (2017) An Advanced Human-Robot Interaction Interface for Teaching Collaborative Robots New Assembly Tasks, ICR 2017: Interactive Collaborative Robotics, pp. 180–190.

Poole H. (1989) Fundamentals of Robotics Engineering, Springer, 436 p.

Robotic Industries Association (2018a) Unimate, the First Industrial Robot. Available at: (accessed 7.9.2018) https://www.robotics.org/joseph-engelberger/unimate.cfm.

Robotic Industries Association (2018b), Robotmaster V7 Exploits the Full Capabilities of Any Robotic Cell, Available at: (accessed 08.10.18) https://www.robotics.org/content-detail.cfm/Industrial-Robotics-News/Robotmaster-V7-Exploits-the-Full-Capabilities-of-Any-Robotic-Cell/content_id/7315.

RobotMaster (2018a), Robotmaster website, Available at: (accessed 08.10.18) https://www.robotmaster.com/en/

RobotMaster (2018b), Robots supported, Available at: (accessed 08.10.18) https://www.robotmaster.com/en/partners

Rosell J., Basañez L., Díaz I. (2002) Graphical task-level robot programming for polishing and grinding, IFAC Proceedings Volume 35, Issue 1, pp. 235–240.

Shimada M., Asakura T. (1996) Interactive Interface of Manipulator with Task Planner, Proceedings of the 1996 IEEE IECON. 22nd International Conference on Industrial Electronics, Control, and Instrumentation, pp. 1227–1232

Shimano B. (1979) VAL: A versatile robot programming and control system, COMPSAC 79. Proceedings. Computer Software and The IEEE Computer Society's Third International Applications Conference, pp. 878–883

Thangaraj A.R, Doelfs M. (1991) Reduce downtime with Off-Line programming, Robotics Today 4(2), pp. 1–3

Wang Y., Ma HS., Yang JH., Wang KS. (2017) Industry 4.0: a way from mass customization to mass personalization production, Advances in Manufacturing Vol. 5 Issue 4, pp. 311–320.

Wassermann J., Vick A., Krüger J. (2018) Intuitive robot programming through environment perception, augmented reality simulation and automated program verification, Procedia CIRP Vol. 76, pp. 161–166.

Yong Y., Bonney M., Off-line programming, in: Nof S. (1999) Handbook of industrial robotics: Second edition, pp. 353–372.

Yong Y., Gleave J., Green J., Bonney M., Off-line programming for robots, in: Nof S. (1985) Handbook of industrial robotics, John Wiley & Sons, pp. 366–380.